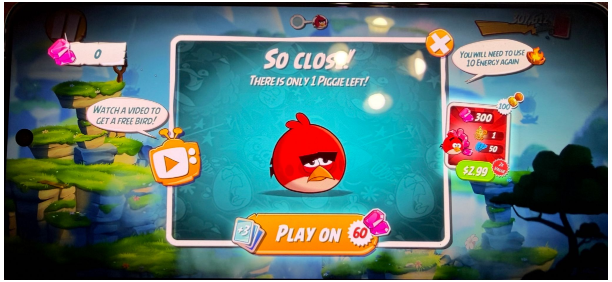
Level Up Your Game: OS Kernel and Game interactions revealed with Perfetto

Ramesh Peri

Agenda

- Perfetto and Game Insights
  - Overall time
  - Frame Time
  - IRQs/SoftIRQs
  - Wakeups
  - Time in OS Scheduler
  - Core Sleep
- Perfetto and Memory Leaks
  - A simple example with memory leak



Traces are collected on Pixel8 running angry birds

Games vs. Apps

	Game	**Normal App**
Repetitiveness	Every Frame	Usually not repetitive
Realtime	Yes	No
Latency Sensitivity	Yes	No
Memory Usage	Spiky	Uniform
Kernel Impact	Kernel scheduler, interrupts,tick rates, migrations, affinity, IRQs	Minimal
Scheduler Quantum	In milliseconds	In seconds

Tools

- **Perfetto**
- Simpleperf
- bpftrace

Perfetto config to collect data

```
data_sources: {
    config {
        name: "linux.ftrace"
        target_buffer: 0
        ftrace_config {
            ftrace_events: "sched/sched_switch"
            ftrace_events: "sched/sched_waking"
            ftrace_events: "sched/sched_wakeup"
            ftrace_events: "power/cpu_frequency"
            ftrace_events: "irq/irq_handler_entry"
            ftrace_events: "irq/irq_handler_exit"
            ftrace_events: "power/cpu_idle"
            ftrace_events: "task/task_rename"
            ftrace_events: "irq/softirq_entry"
            ftrace_events: "irq/softirq_exit"
            ftrace_events: "irq/softirq_raise"
        }
    }
}
```

OS scheduler events

irq/softirq entry exits

The overhead is reasonably low with these events - around 2-3%

Overall Performance

10 secs of wall time across 9 cores

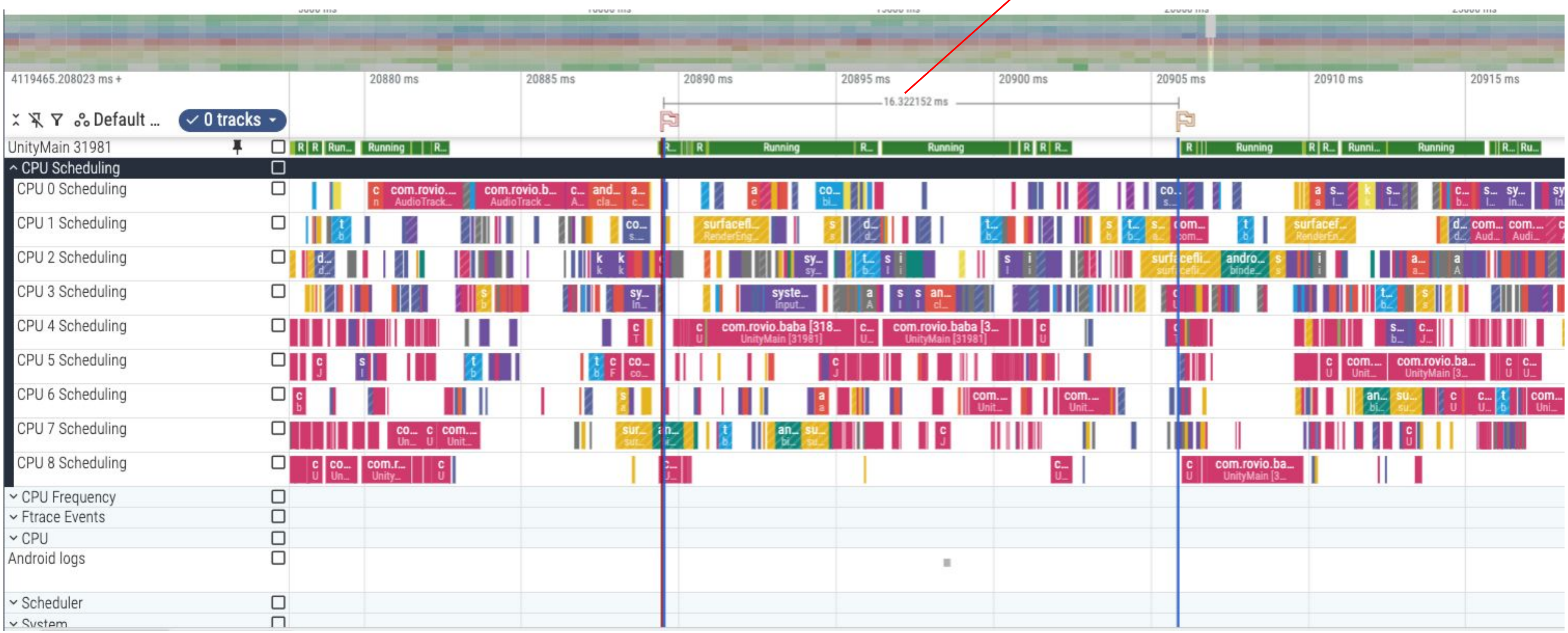
Area Selection   CPU by thread   CPU by process

Process	PID	Wall duration	Wall duration %	Avg Wall duration	Occurrences
		Σ 41.48s			Σ 232828
com.rovio.baba	31829	17.36s	41.86%	146.2µs	118755
/system/bin/surfaceflinger	604	5.054s	12.18%	313.6µs	16114
/system/bin/traced_probes	1130	2.229s	5.37%	1.207ms	1846
/vendor/bin/hw/android.hardware.composer.hwc3-service.pixel	607	1.605s	3.87%	443.1µs	3623
/system/bin/audioserver	990	1.467s	3.54%	146.8µs	9993
com.google.android.gms	14085	1.413s	3.41%	421.9µs	3349
system_server	1441	1.394s	3.36%	302.3µs	4613
/vendor/bin/hw/android.hardware.audio.service	916	1.326s	3.20%	125.1µs	10602
com.google.android.apps.photos	1394	1.167s	2.81%	190.7µs	6118

41.48sec out of 90 secs - 46% utilization

Frame time

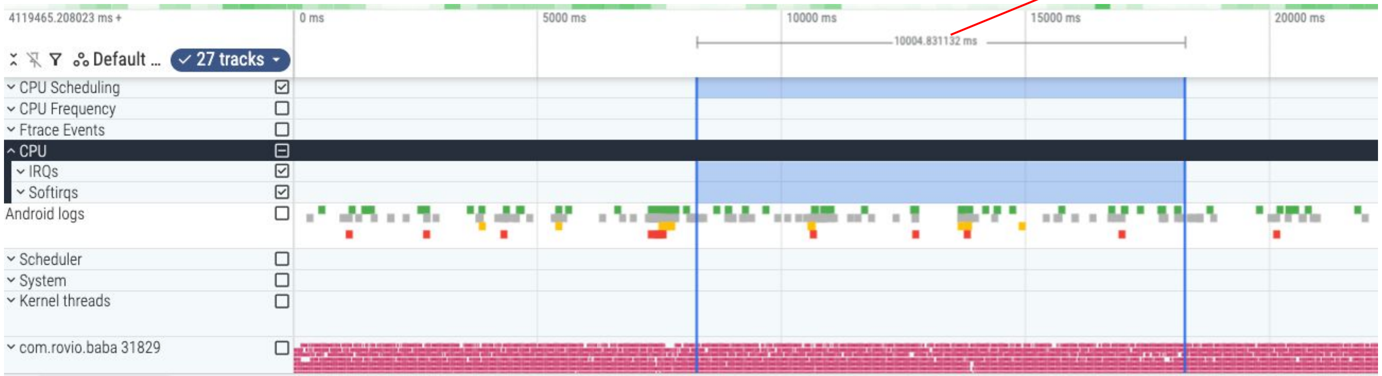
Here 16.6 ms which
is 60FPS



Main process com.rovio.baba's UnityMain thread shows the frame time

IRQs/softIRQs and its impact

41.48 secs of cpu time



Current Selection | Android Logs | Ftrace Events

Area Selection | Slices | Pivot Table | Slice Flamegraph | CPU by thread | CPU by process

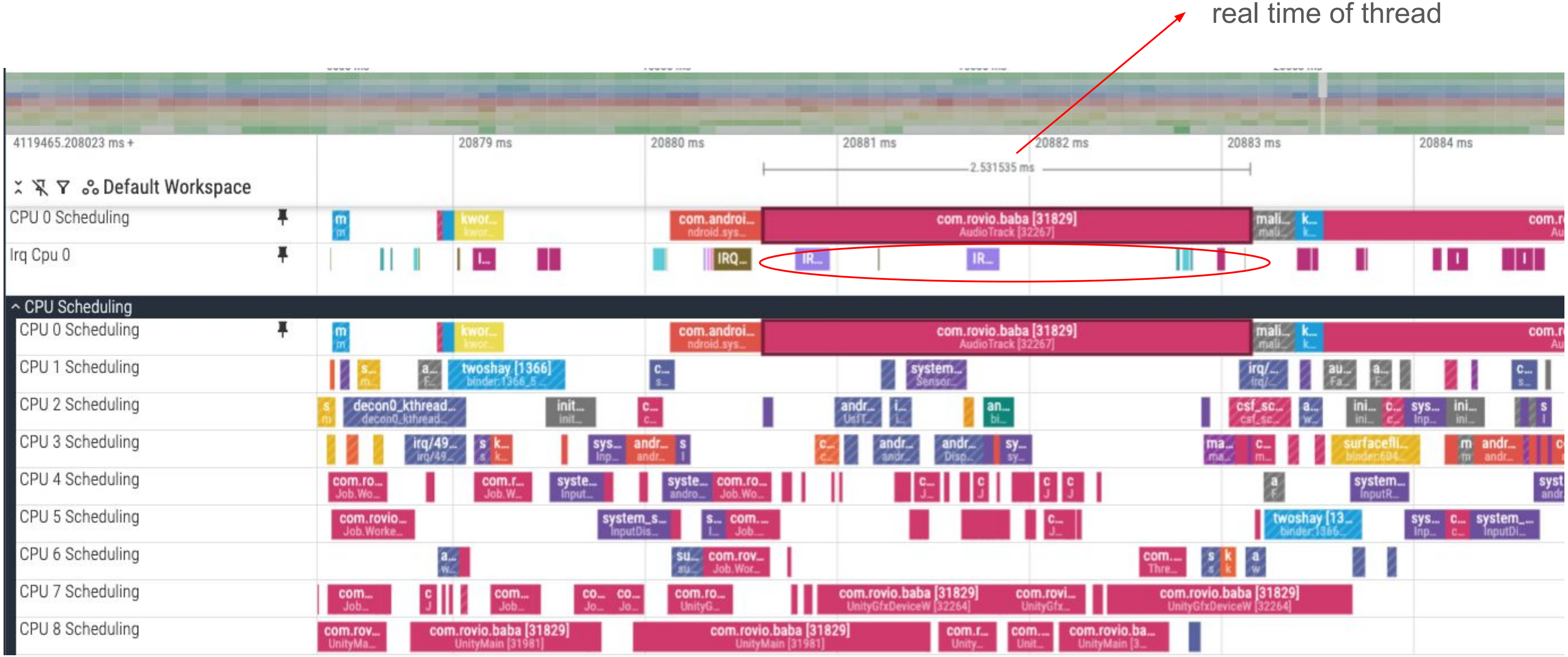
name	sum(dur)	count
Total values:	4058.816728 ms	242368
IRQ (IPI)	453.290311 ms	151038
IRQ (arch_timer)	1062.857139 ms	29655
IRQ (1f000000.mali)	925.151327 ms	15552
SCHED	250.82002 ms	12806
RCU	60.630197 ms	12770
TIMER	276.657566 ms	6800
IRQ (ufshcd)	63.839008 ms	2603
BLOCK	189.515135 ms	2459
IRQ (19470000.drmdecon)	200.299266 ms	1779
IRQ (15220000.mbox)	195.295128 ms	1001

4.05 secs in IRQs and softIRQs

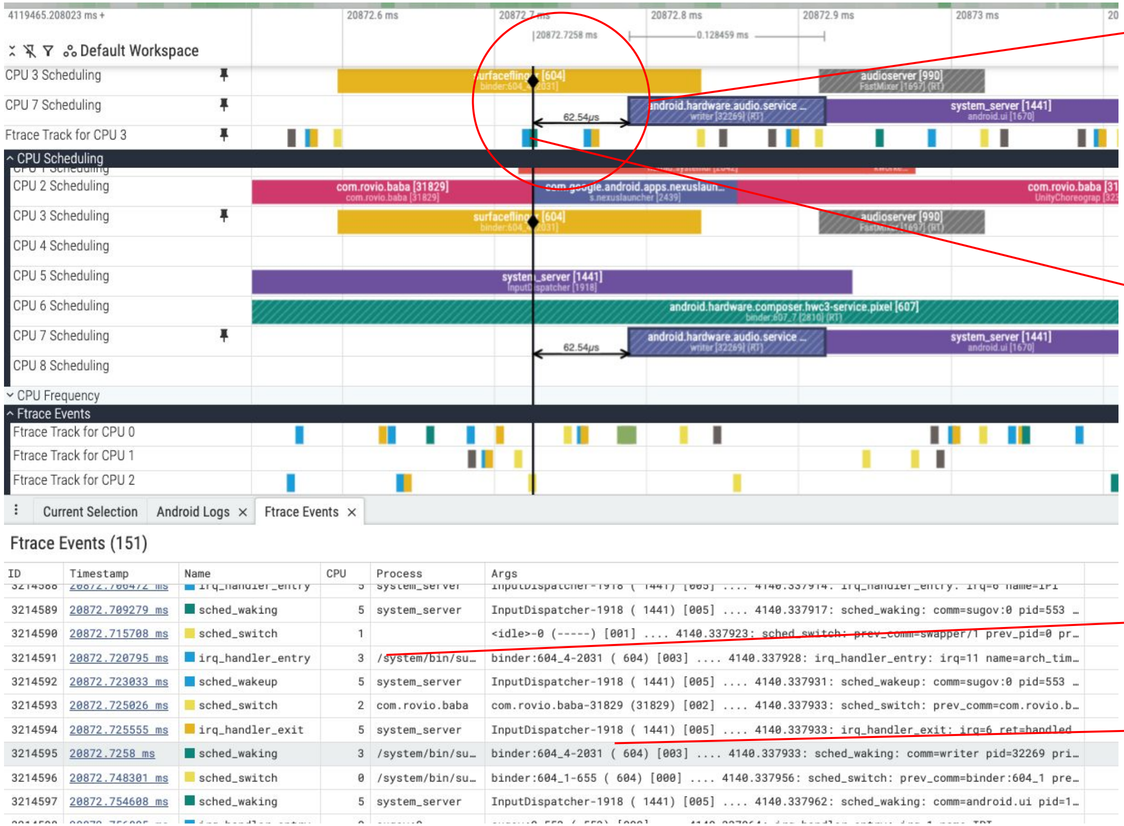
Total time on user threads is 41.48-4.05 = 37.43 secs

IRQs and Thread timings

Subtract the times of interrupt handlers to get the real time of thread



Thread wakeups and their reasons



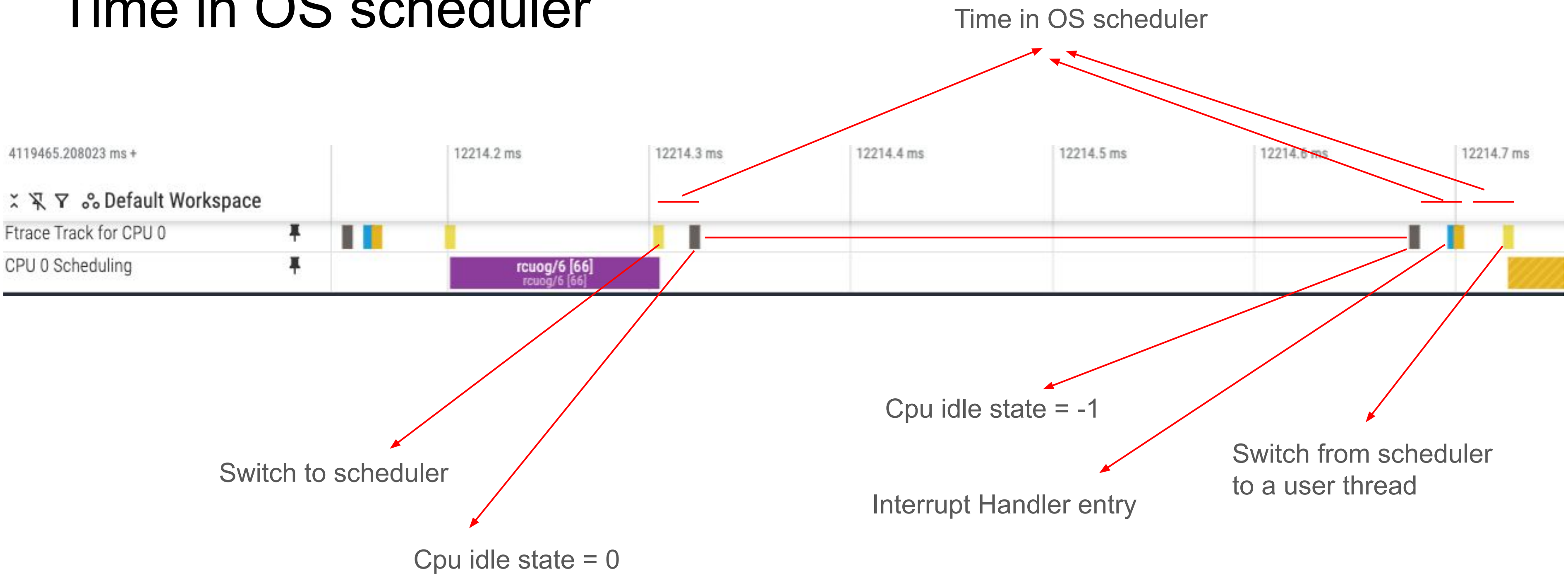
Audio thread woken by surfaceflinger

In reality audio thread woken by arch timer interrupt

cpu3 in interrupt handler

Interrupt handler wakes up audio writer thread

Time in OS scheduler



OS scheduler time is distributed across many small time slots and can be up to 15% of total execution time

Total Sleep time where cores are really idle

Sum up all the time between cpu idle state=0 and cpu idle state=-1 for each core to get the total sleep time

More Perfetto

- The perfetto trace file format is well documented
- Can use trace processor and sql queries to get deeper info than what UI can provide
- Can use python trace processor bindings to write advanced metric calculations that UI cannot provide

Extremely power tool to provide deep insights into how OS and hw is being used

Memory

- Memory Leaks are a problem in games on due to limited amount of memory on the devices
- The app needs to be profileable for this technique to work

Perfetto config to collect heap snapshots

```
data_sources: {
  config {
    name: "android.heapprofd"
    target_buffer: 0
    heapprofd_config {
      sampling_interval_bytes: 1
      shmem_size_bytes: 8388608
      continuous_dump_config {
         dump_phase_ms:1000
         dump_interval_ms:1000
      }
      block_client: true
      process_cmdline: "meamleak"
    }
  }
}
```

Initial dump interval

Dump interval

Name of Process

An Example

```c
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
float *a,*b,*c,*d;
int main(int argc, char **argv) {
   int x;
   printf("press key to start\n");
   scanf("%d",&x);
   if (argc !=2) {
      printf("USAGE: %s <size>\n",argv[0]);
      exit(1);
  }
   int size = 1 << atoi(argv[1]);
   a = (float *)malloc(size*sizeof(float));
   while (1) {
      b = (float *)malloc(size*sizeof(float));
      c = (float *)malloc(size*sizeof(float));
      d = (float *)malloc(size*sizeof(float));
      printf("%p,%p,%p,%p\n",a,b,c,d);
      // touch every 1kb
      for (unsigned int i=0;i<size;i+=1024) {
         a[i]=i&0x34;
         b[i]=(i-56)&0xdeadbeef;
         c[i]=(i+8)&0xdead;
         d[i]=(i+4563)&0xbeef;
      }
         free(c);
         free(d);
   }
   free(a);
   printf("%f\n",a[2048]);
   scanf("%d",&x);
   return(0);
}
```

b is leaking since it is allocated but not freed

Perfetto trace collection process

- Run the program as
  - Memleak 8
- In another window run perfetto after setting the runtime to be 20sec and dump interval to be 1sec
- Run the heap profiling collection
- Load the trace into the perfetto viewer

Perfetto trace

This callsite continuously keep increasing for unreleased malloc size

UI is not very intuitive to see the leaks immediately

Tables in UI

Information is present here.

- Heap_profile_allocation has all the allocations
- Stack_profile_callsite has the callsite information
- Stack_profile_frame has the frame information
- Stack_profile_mapping
- Clock_snap_shot - types of clocks in the system
- Stack_profile_symbol - name of the symbol for the frame

Sql queries

- We want to know the call sites where there are no frees across all the snapshots
- All the call sites which return a +ve number are potential candidates for leaks

select callsite_id, min(count) as m from heap_profile_allocation group by callsite_id order by m desc

```
> select callsite_id, min(count) as m from heap_profile_allocation group by callsite_id order by m desc
callsite_id          m
-------------------- --------------------
                   2                24800
                   6               -48000
                   4               -48000

Query executed in 0.827 ms
```

Callsite 4 which is NOT leaking memory

```
> select *,(ts-(select min(ts) from heap_profile_allocation))/1000000000 from heap_profile_allocation where callsite_id=4
```

id	type	ts	upid	heap_name	callsite_id	count	size	(ts-(select min(ts)
1	heap_profile_allocat	34744546253856	731	libc.malloc	4	24800	25395200	0
2	heap_profile_allocat	34744546253856	731	libc.malloc	4	-24800	-25395200	0
6	heap_profile_allocat	34745546253956	731	libc.malloc	4	48000	49152000	1
7	heap_profile_allocat	34745546253956	731	libc.malloc	4	-48000	-49152000	1
11	heap_profile_allocat	34746549587389	731	libc.malloc	4	47200	48332800	2
12	heap_profile_allocat	34746549587389	731	libc.malloc	4	-47200	-48332800	2
16	heap_profile_allocat	34747552920822	731	libc.malloc	4	42456	43474944	3
17	heap_profile_allocat	34747552920822	731	libc.malloc	4	-42455	-43473920	3
21	heap_profile_allocat	34748556254256	731	libc.malloc	4	42646	43669504	4
22	heap_profile_allocat	34748556254256	731	libc.malloc	4	-42646	-43669504	4
26	heap_profile_allocat	34749559587689	731	libc.malloc	4	47400	48537600	5
27	heap_profile_allocat	34749559587689	731	libc.malloc	4	-47400	-48537600	5
31	heap_profile_allocat	34750559587789	731	libc.malloc	4	47400	48537600	6
32	heap_profile_allocat	34750559587789	731	libc.malloc	4	-47400	-48537600	6
36	heap_profile_allocat	34751562921222	731	libc.malloc	4	47000	48128000	7
37	heap_profile_allocat	34751562921222	731	libc.malloc	4	-47000	-48128000	7
41	heap_profile_allocat	34752562921322	731	libc.malloc	4	38169	39085056	8
42	heap_profile_allocat	34752562921322	731	libc.malloc	4	-38169	-39085056	8
46	heap_profile_allocat	34753569588088	731	libc.malloc	4	37898	38807552	9
47	heap_profile_allocat	34753569588088	731	libc.malloc	4	-37898	-38807552	9
51	heap_profile_allocat	34754572921522	731	libc.malloc	4	45800	46899200	10
52	heap_profile_allocat	34754572921522	731	libc.malloc	4	-45800	-46899200	10
56	heap_profile_allocat	34755576254955	731	libc.malloc	4	47600	48742400	11
57	heap_profile_allocat	34755576254955	731	libc.malloc	4	-47600	-48742400	11
61	heap_profile_allocat	34756559588386	731	libc.malloc	4	46742	47863808	12
62	heap_profile_allocat	34756559588386	731	libc.malloc	4	-46742	-47863808	12
66	heap_profile_allocat	34757576255154	731	libc.malloc	4	44000	45056000	13
67	heap_profile_allocat	34757576255154	731	libc.malloc	4	-44000	-45056000	13
71	heap_profile_allocat	34758582921921	731	libc.malloc	4	47200	48332800	14
72	heap_profile_allocat	34758582921921	731	libc.malloc	4	-47200	-48332800	14
76	heap_profile_allocat	34759582922021	731	libc.malloc	4	41124	42110976	15
77	heap_profile_allocat	34759582922021	731	libc.malloc	4	-41124	-42110976	15

...

+ve is the number of allocations and -ve is number of frees

Callsite 2 which is leaking memory

```
> select *,(ts-(select min(ts) from heap_profile_allocation))/1000000000 from heap_profile_allocation where callsite_id=2
```

id	type	ts	upid	heap_name	callsite_id	count	size	(ts-(select min(ts)
0	heap_profile_allocat	34744546253856	731	libc.malloc	2	24800	25395200	0
5	heap_profile_allocat	34745546253956	731	libc.malloc	2	48000	49152000	1
10	heap_profile_allocat	34746549587389	731	libc.malloc	2	47200	48332800	2
15	heap_profile_allocat	34747552920822	731	libc.malloc	2	42456	43474944	3
20	heap_profile_allocat	34748556254256	731	libc.malloc	2	42646	43669504	4
25	heap_profile_allocat	34749559587689	731	libc.malloc	2	47400	48537600	5
30	heap_profile_allocat	34750559587789	731	libc.malloc	2	47400	48537600	6
35	heap_profile_allocat	34751562921222	731	libc.malloc	2	47000	48128000	7
40	heap_profile_allocat	34752562921322	731	libc.malloc	2	38169	39085056	8
45	heap_profile_allocat	34753569588088	731	libc.malloc	2	37898	38807552	9
50	heap_profile_allocat	34754572921522	731	libc.malloc	2	45800	46899200	10
55	heap_profile_allocat	34755576254955	731	libc.malloc	2	47600	48742400	11
60	heap_profile_allocat	34756559588386	731	libc.malloc	2	46742	47863808	12
65	heap_profile_allocat	34757576255154	731	libc.malloc	2	44000	45056000	13
70	heap_profile_allocat	34758582921921	731	libc.malloc	2	47200	48332800	14
75	heap_profile_allocat	34759582922021	731	libc.malloc	2	41124	42110976	15
80	heap_profile_allocat	34760589588788	731	libc.malloc	2	47400	48537600	16
85	heap_profile_allocat	34761589588887	731	libc.malloc	2	47200	48332800	17
90	heap_profile_allocat	34762522922314	731	libc.malloc	2	43669	44717056	17

Query executed in 0.769 ms

+ve is the number of allocations and no -ve means there are NO frees

Tables to help get Symbolized callstacks

```
> select * from stack_profile_frame
```

id	type	name	mapping	rel_pc	symbol_set_id	deobfuscated_name
0	stack_profile_frame	__libc_init	0	573224	[NULL]	[NULL]
1	stack_profile_frame	main	1	18756	[NULL]	[NULL]
2	stack_profile_frame	malloc	0	325100	[NULL]	[NULL]
3	stack_profile_frame	main	1	18776	[NULL]	[NULL]
4	stack_profile_frame	main	1	18792	[NULL]	[NULL]

Query executed in 0.738 ms

```
> select * from stack_profile_callsite
```

id	type	depth	parent_id	frame_id
0	stack_profile_callsi	0	[NULL]	0
1	stack_profile_callsi	1	0	1
2	stack_profile_callsi	2	1	2
3	stack_profile_callsi	1	0	3
4	stack_profile_callsi	2	3	2
5	stack_profile_callsi	1	0	4
6	stack_profile_callsi	2	5	2

Query executed in 0.336 ms

Questions